Implementing MLPs with Keras

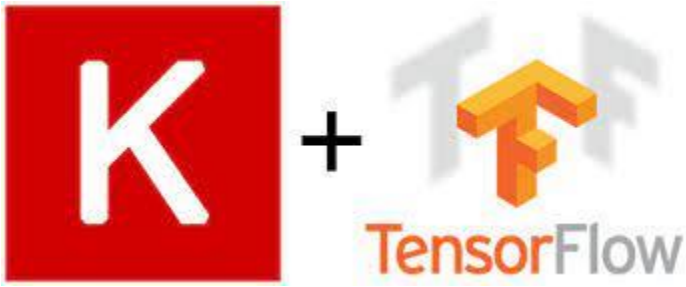
Chapter 10

Today

- Building an Image Classifier Using the Sequential API.
- Building a Regression MLP Using the Sequential API.

Implementing MLPs with Keras

- Keras is TensorFlow's high-level deep learning API.
- It allows you to build, train, evaluate, and execute all sorts of neural networks.
- Keras used to support multiple backends, including TensorFlow, PlaidML, Theano, and Microsoft Cognitive Toolkit (CNTK), but since version 2.4, Keras is **TensorFlow-only**



Building an MLP for Image Classification

- Data set:
  - Fashion MNIST,
  - It has (70,000) grayscale images of 28 × 28 pixels each, with 10 classes), the images represent fashion items rather
  - A simple linear model reaches about 92% accuracy on Digits MNIST, but only about 83% on Fashion MNIST.

Building an MLP for Image Classification

- How to create the model(ANN) using the sequential API:
  1. Build the neural network.
  2. Compile the neural network.
  3. Training and evaluating the neural network.

Building an MLP for Image Classification

- Let's load Fashion MNIST. It's already shuffled and split into a training set (60,000 images) and a test set (10,000 images), but we'll hold out the last 5,000 images from the training set for validation:

```
import tensorflow as tf

fashion_mnist = tf.keras.datasets.fashion_mnist.load_data()
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist
X_train, y_train = X_train_full[:-5000], y_train_full[:-5000]
X_valid, y_valid = X_train_full[-5000:], y_train_full[-5000:]
```

Building an MLP for Image Classification

```
>>> X_train.shape
(55000, 28, 28)
>>> X_train.dtype
dtype('uint8')
```

```
X_train, X_valid, X_test = X_train / 255., X_valid / 255., X_test / 255.
```

Building an MLP for Image Classification

- To build the neural network:

```
tf.random.set_seed(42)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=[28, 28]))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(300, activation="relu"))
model.add(tf.keras.layers.Dense(100, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="softmax"))
```

Sequential model. This is the simplest kind of Keras model for neural networks that are just composed of a single stack of layers connected sequentially. This is called the sequential API.

OR

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

Building an MLP for Image Classification

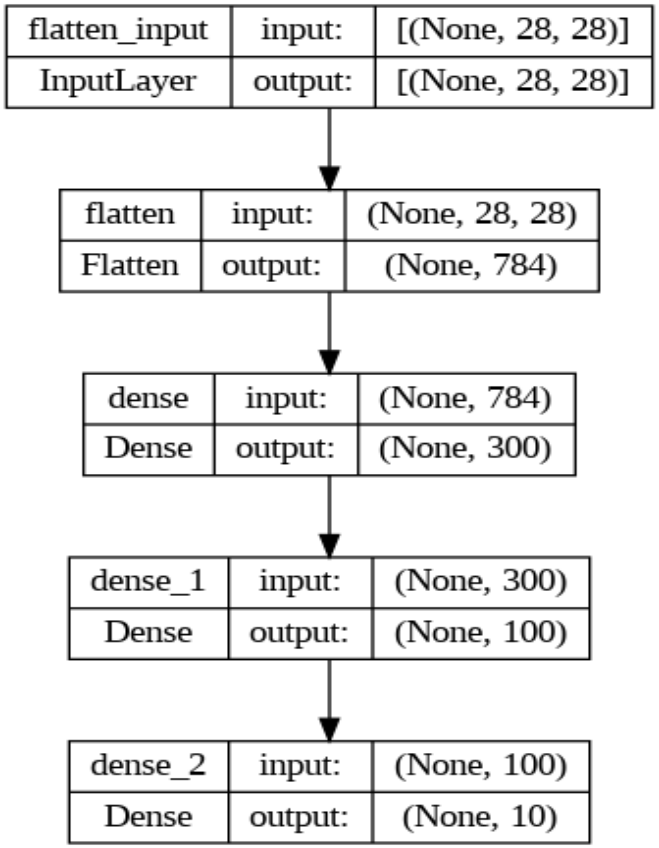
```
>>> model.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
```



Compiling the model

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

To measure the accuracy during training and evaluation, which is why we set metrics=["accuracy"].

If your Yi's are one-hot encoded, use categorical_crossentropy. Examples (for a 3-class classification): [1,0,0] , [0,1,0], [0,0,1]

But if your Yi's are integers, use sparse_categorical_crossentropy. Examples for above 3-class classification problem: [1] , [2], [3]

Training and Evaluating the Model

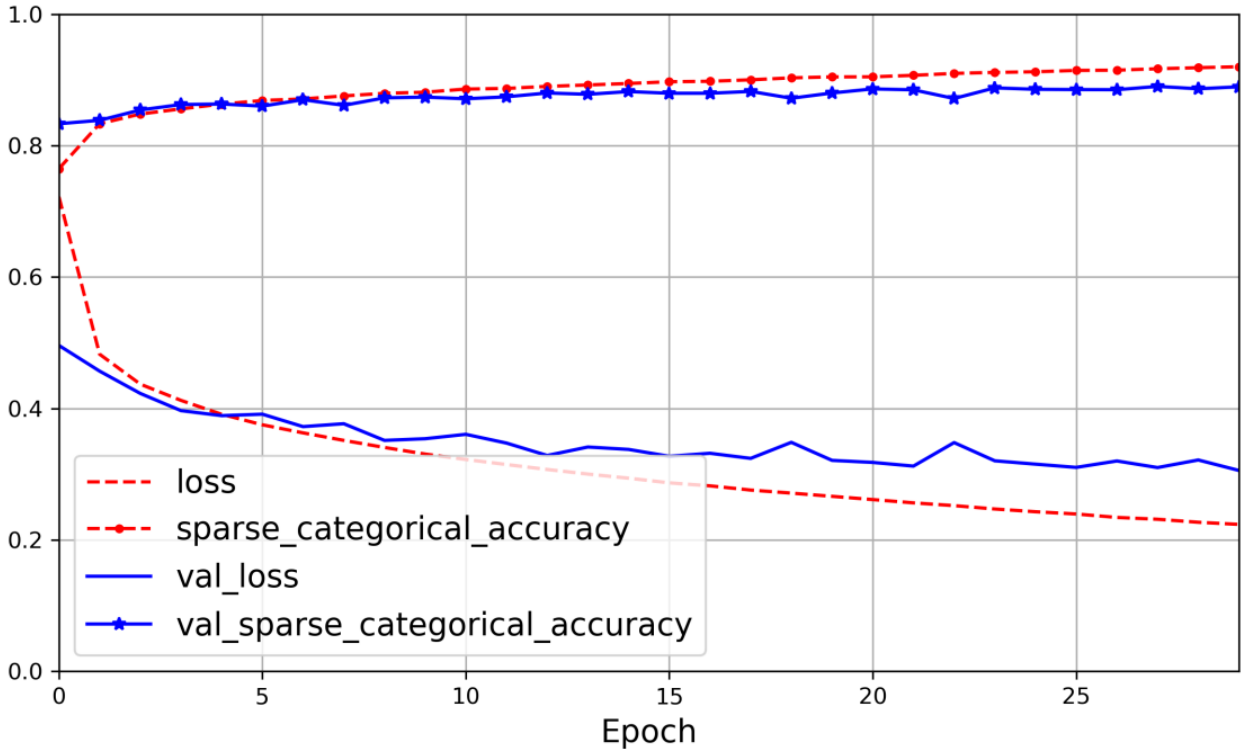
```
>>> history = model.fit(X_train, y_train, epochs=30,
...                     validation_data=(X_valid, y_valid))
...
Epoch 1/30
1719/1719 [==============================] - 2s 989us/step
  - loss: 0.7220 - sparse_categorical_accuracy: 0.7649
  - val_loss: 0.4959 - val_sparse_categorical_accuracy: 0.8332
Epoch 2/30
1719/1719 [==============================] - 2s 964us/step
  - loss: 0.4825 - sparse_categorical_accuracy: 0.8332
  - val_loss: 0.4567 - val_sparse_categorical_accuracy: 0.8384
[...]
Epoch 30/30
1719/1719 [==============================] - 2s 963us/step
  - loss: 0.2235 - sparse_categorical_accuracy: 0.9200
  - val_loss: 0.3056 - val_sparse_categorical_accuracy: 0.8894
```

The fit() method returns a History object containing:

- The training parameters (history.params).
- The list of epochs it went through (history.epoch),
- A dictionary (history.history) containing the loss and extra metrics it measured at the end of each epoch on the training set and on the validation set (if any)

Training and Evaluating the Model

- Learning curves: the mean training loss and accuracy measured over each epoch, and the mean validation loss and accuracy measured at the end of each epoch



Training and Evaluating the Model

```
>>> model.evaluate(X_test, y_test)
313/313 [==============================] - 0s 626us/step
  - loss: 0.3243 - sparse_categorical_accuracy: 0.8864
[0.32431697845458984, 0.8863999843597412]
```

Evaluate

To make predictions:

```
>>> X_new = X_test[:3]
>>> y_proba = model.predict(X_new)
>>> y_proba.round(2)
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.02, 0.  , 0.97],
       [0.  , 0.  , 0.99, 0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
      dtype=float32)
```

```
>>> import numpy as np
>>> y_pred = y_proba.argmax(axis=-1)
>>> y_pred
array([9, 2, 1])
>>> np.array(class_names)[y_pred]
array(['Ankle boot', 'Pullover', 'Trouser'], dtype='<U11')
```

Prove your answer

```
>>> y_new = y_test[:3]
>>> y_new
array([9, 2, 1], dtype=uint8)
```

Building a Regression MLP Using the Sequential API

```
tf.random.set_seed(42)
norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
model = tf.keras.Sequential([
    norm_layer,
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(1)
])
```

Build

Compile

```
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(loss="mse", optimizer=optimizer, metrics=["RootMeanSquaredError"])
norm_layer.adapt(X_train)
```

```
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
```

Train

Evaluate and predict

```
mse_test, rmse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3]
y_pred = model.predict(X_new)
```

Building a Regression MLP Using the Sequential API

- **Adam optimizer** is the extended version of stochastic gradient descent which could be implemented in various deep learning applications such as computer vision and natural language processing in the future years.
- Adam was first introduced in 2014.
- The output layer has a single neuron and it uses no activation function.
- The loss function is the mean squared error, the metric is the RMSE

Building a Regression MLP Using the Sequential API

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_2 (Normalizat  (None, 8)                17
 ion)

 dense_4 (Dense)             (None, 50)                450

 dense_5 (Dense)             (None, 50)                2550

 dense_6 (Dense)             (None, 50)                2550

 dense_7 (Dense)             (None, 1)                 51

=================================================================
Total params: 5,618
Trainable params: 5,601
Non-trainable params: 17
_________________________________________________________________
```